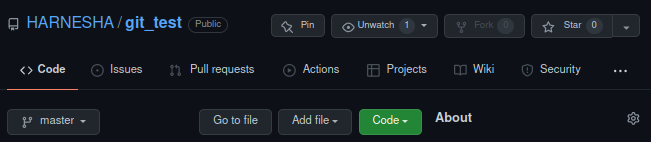
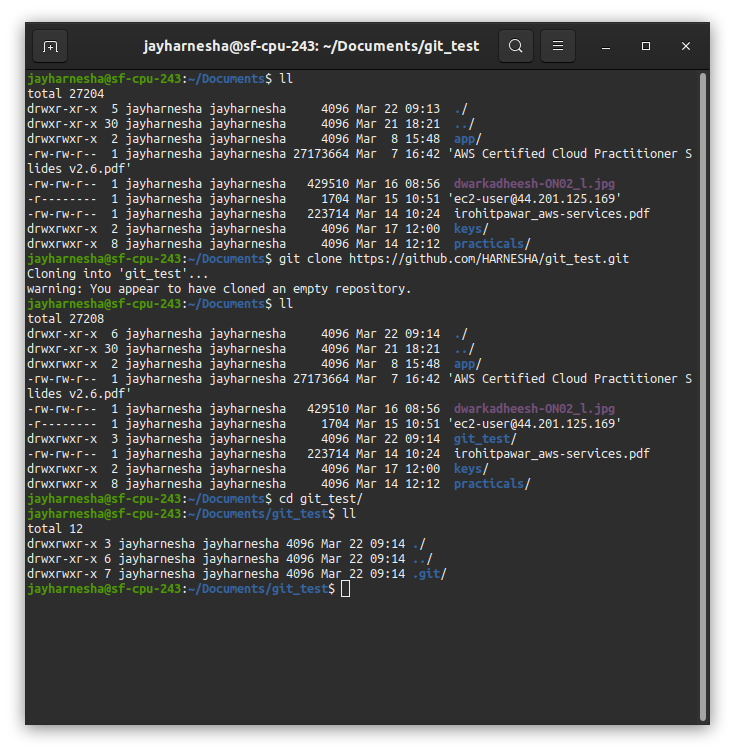
Create git-hub repo.

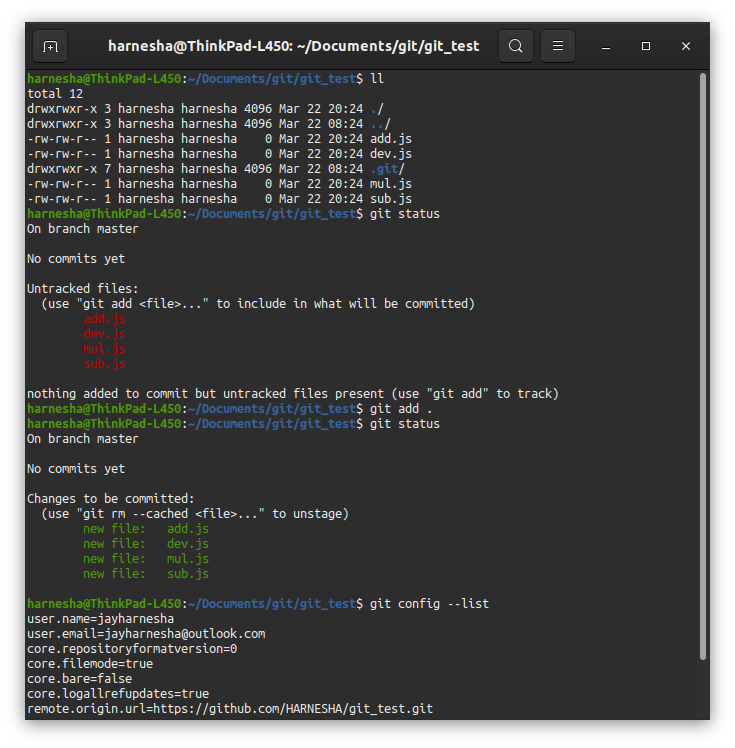


Clone git repo using the following command.

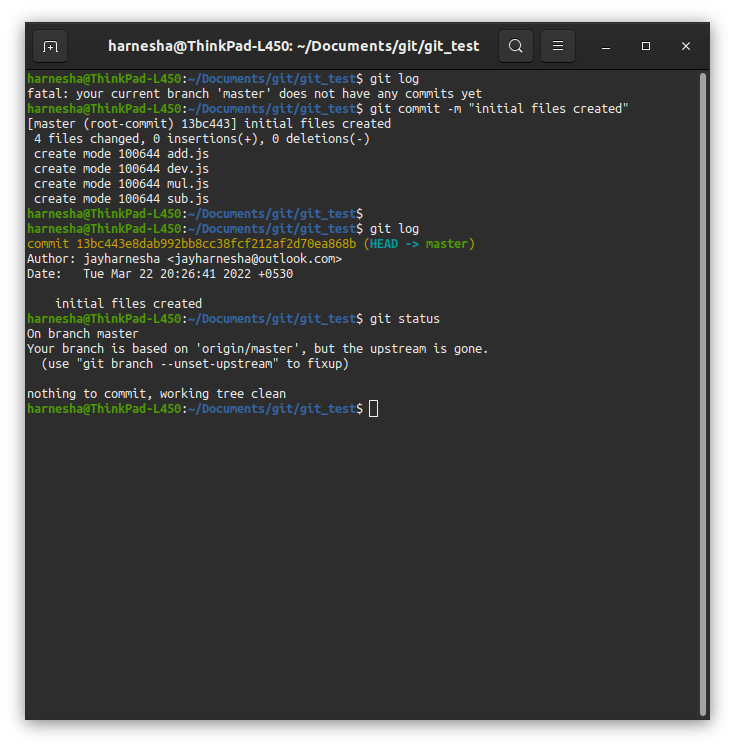
Git clone git-repo



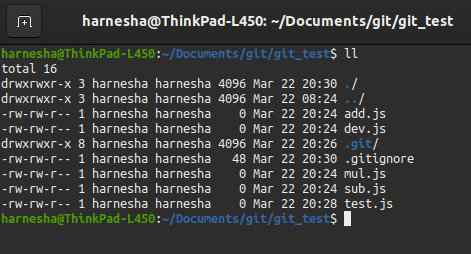
Added four files to git local repo and staged and verify status.

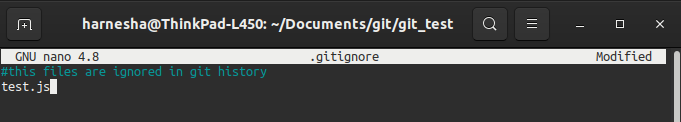


Verify staging status and commit the fiiles to local git repo.

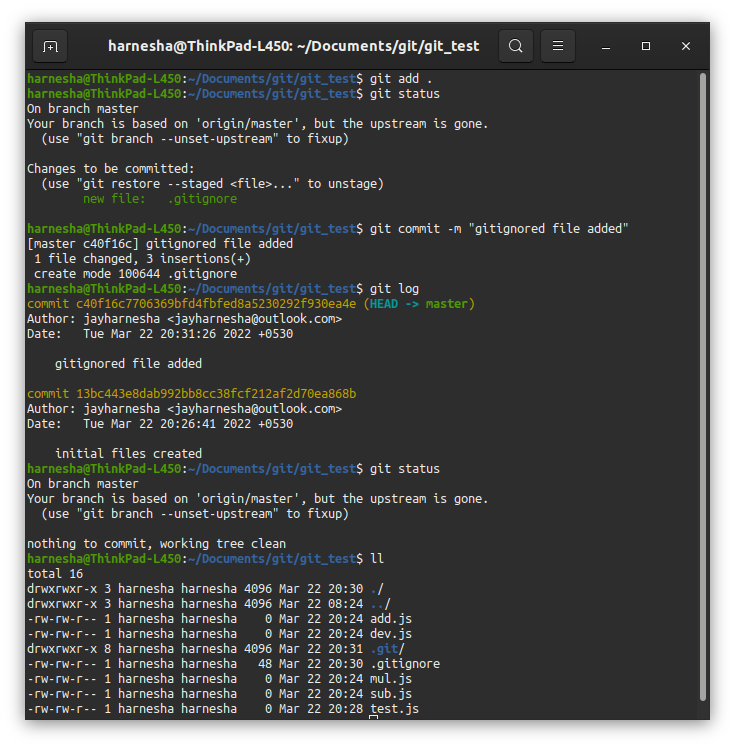


Create .gitignore file if we have to untract any file from out repo and add file name to .gitignore file.

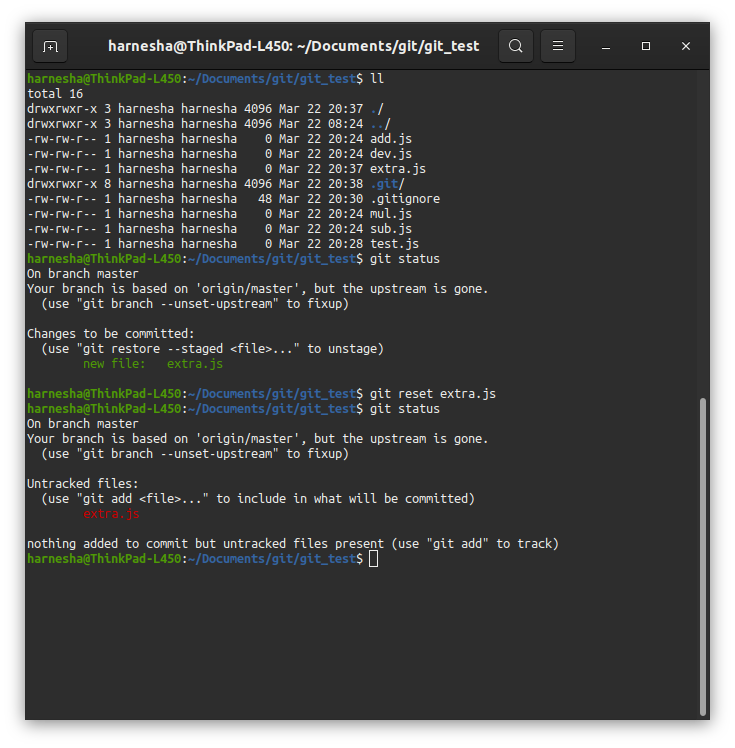




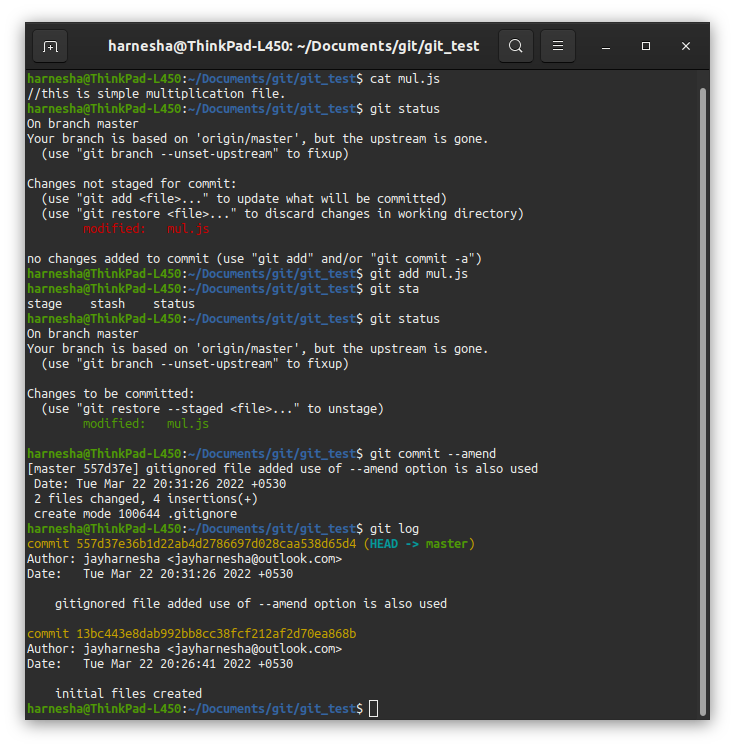
We can see that after creating the file test.js though file is no longer to be trcked in git repo.



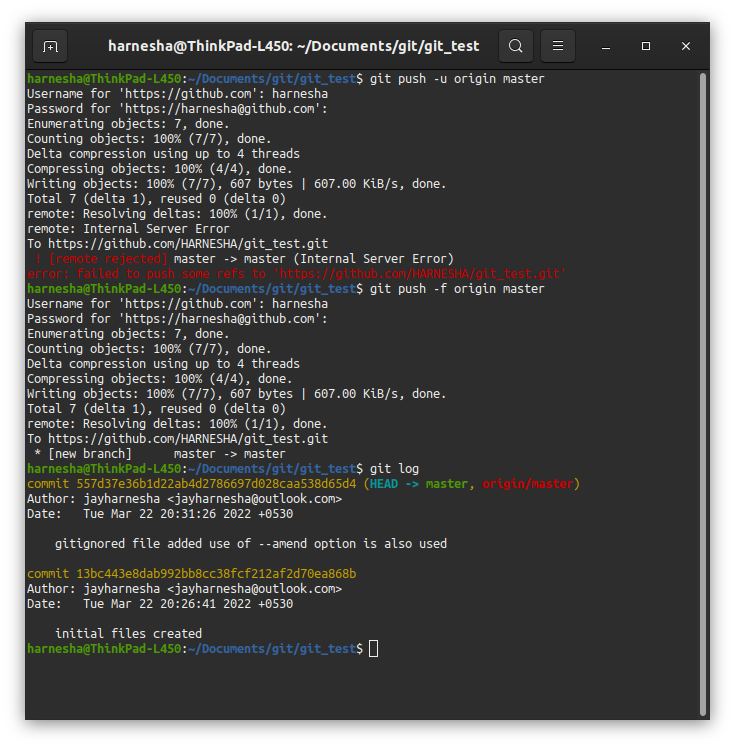
We can upstage the staged file using git reset command and file will be added to our working area.



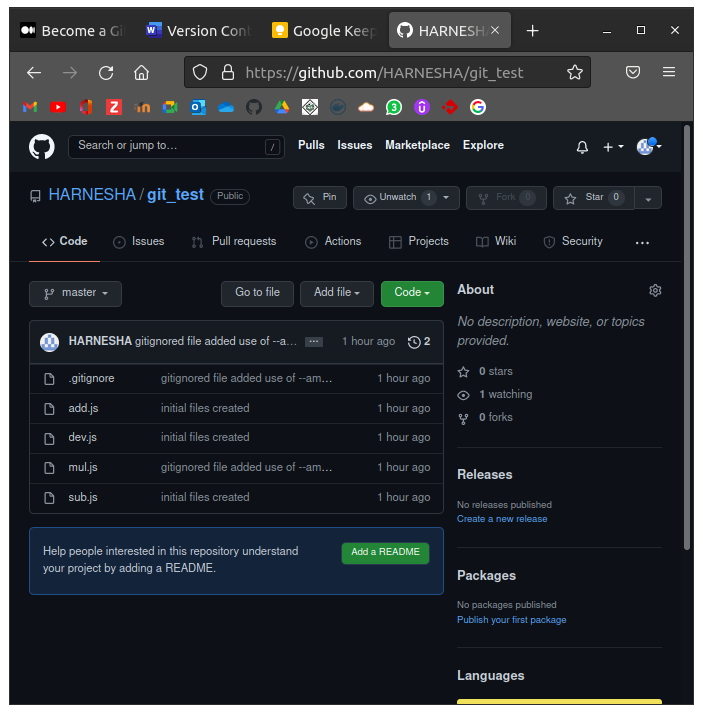
Sometimes, we can make mistakes while writing a commit message. I made a deliberate mistake in the above commit message. To change the message of the previous commit, use the following command.



our commit is located in the local repository and the remote repository has no idea of its existence. Hence we need to push that commit to the remote repository

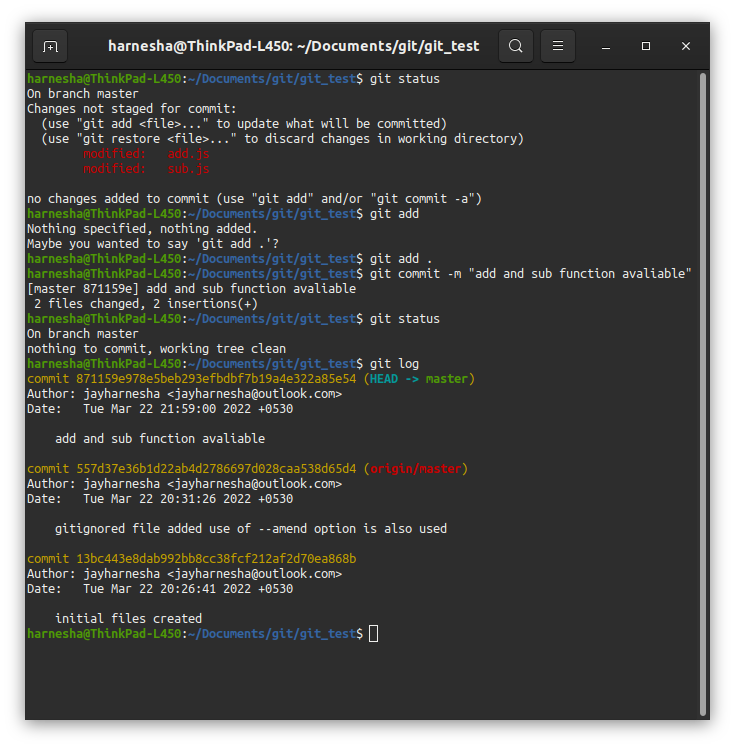


Verify data at master branch in github repo.

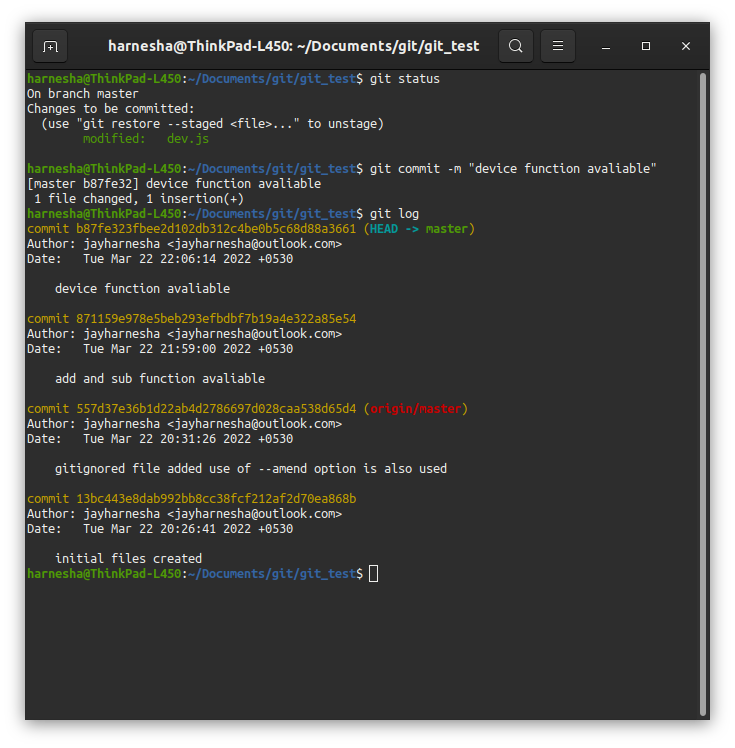


It is possible that you made a commit that you didn’t intend to make. So, what about that? HEAD is pointer to the last commit in Git history.

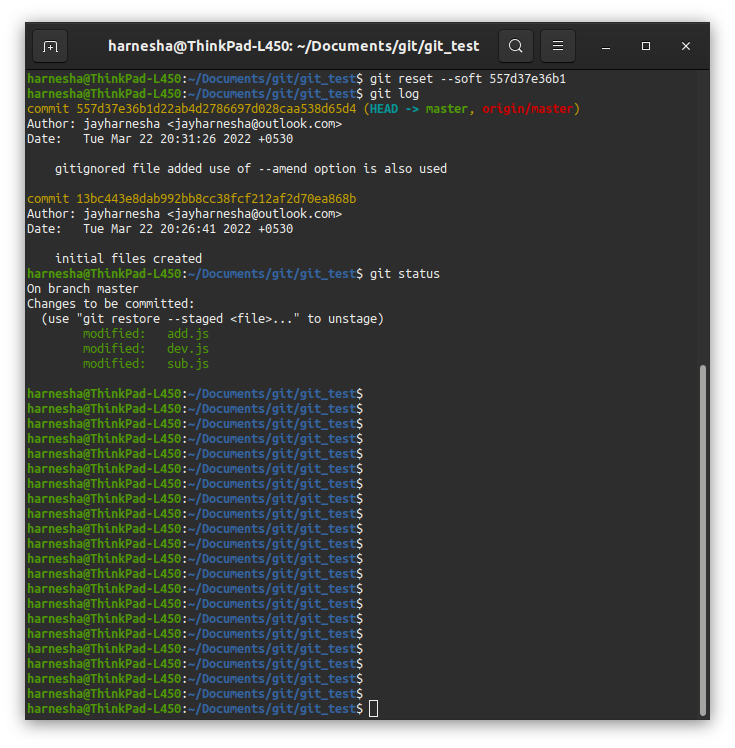
Let’s make some changes in the code and create a new commit.



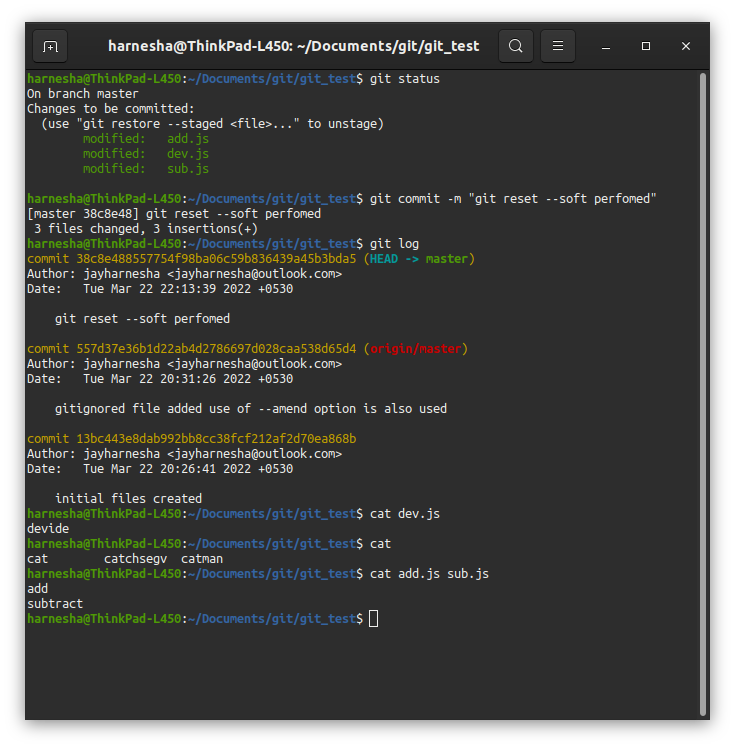
Now our Git history looks like this.



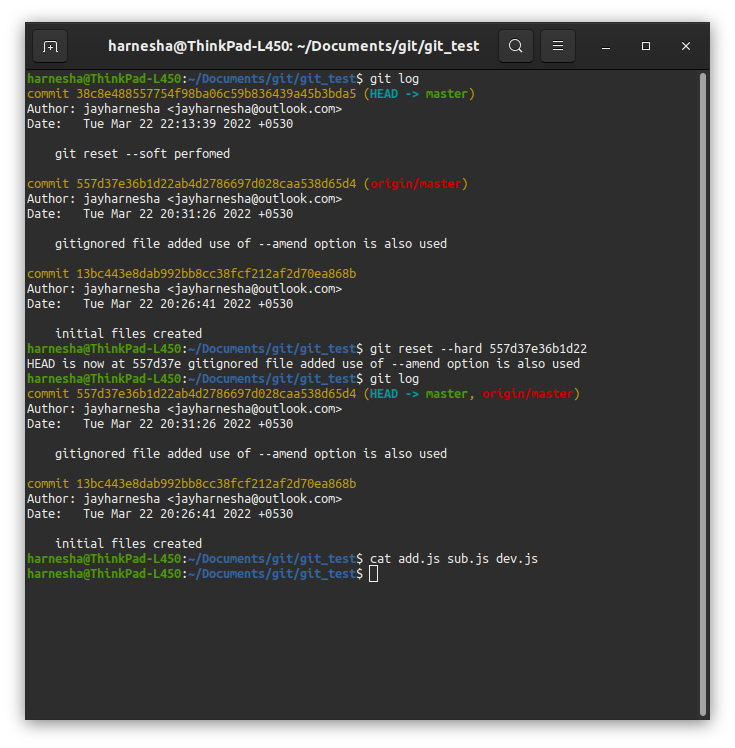
git reset --soft will remove all commits after commit specified commit and will bring all changed code after that into the staging area. You don’t need to use the full hash of a commit. All commits after this commit are then removed from git history.



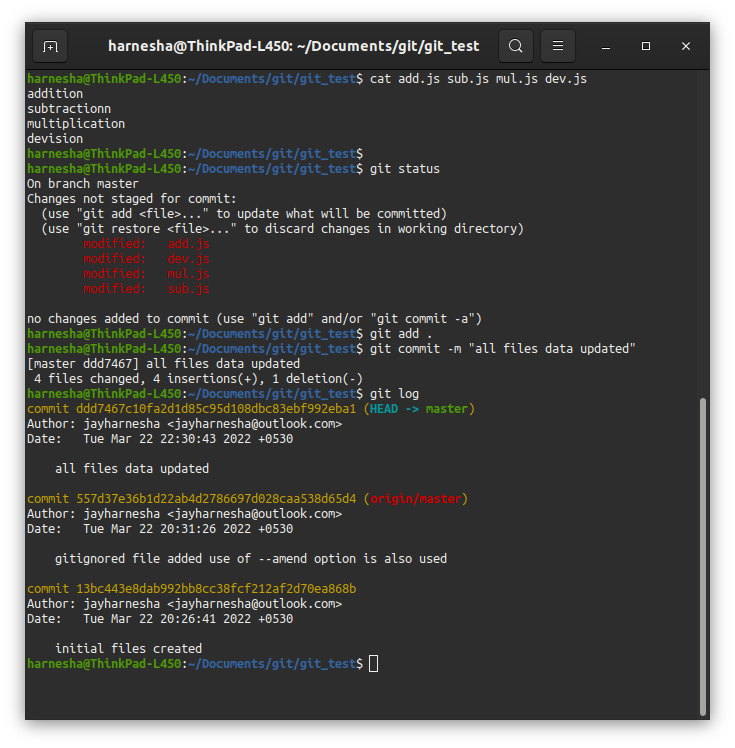
Verify status by git log and git status.



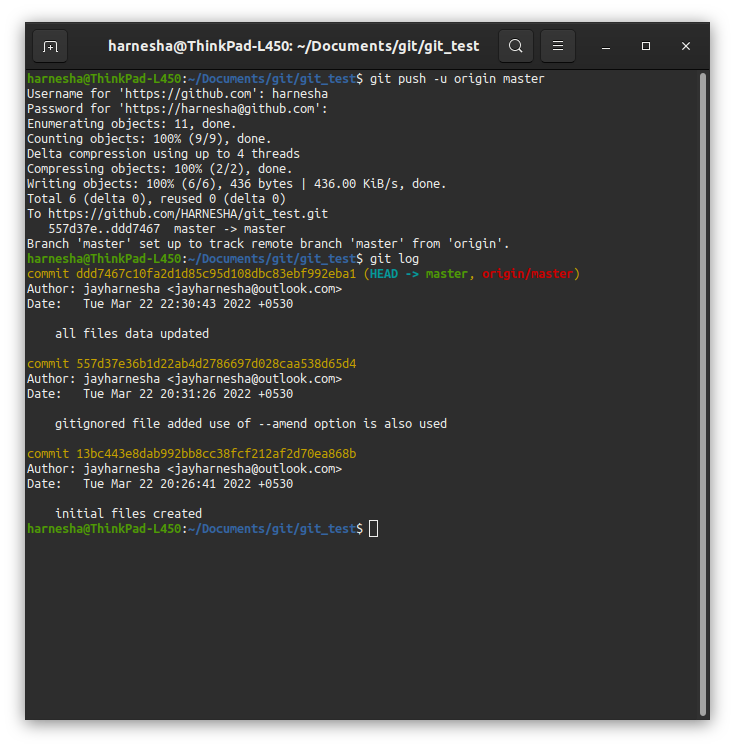
git reset --hard will remove all commits after specific commit and destroy all changed code after that. This will also remove the changed files in the working or staging area. Hence git reset --hard HEAD is also used to get rid of all the changes whether it is inside the working area or the staging area. **One important thing to remember is that all untracked files (*newly created files*) will not be removed.**



You can verify status and see file data details.

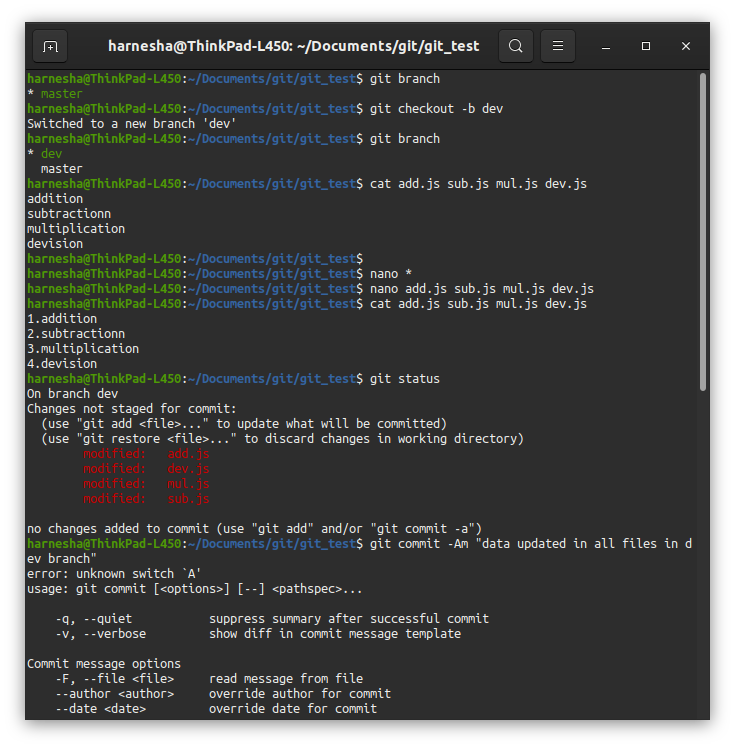


Update the repositry and use git push to update git-hub repositery.



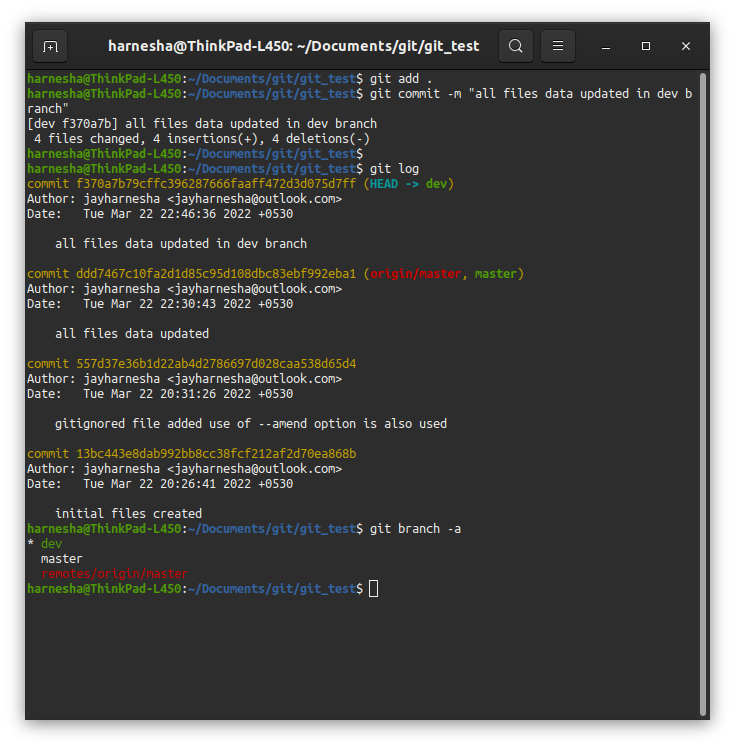
Until now we have been dealing solely with the master branch as we saw in some commands. But what is a branch? Git is all about commits. At any point in time, we are always in some branch.

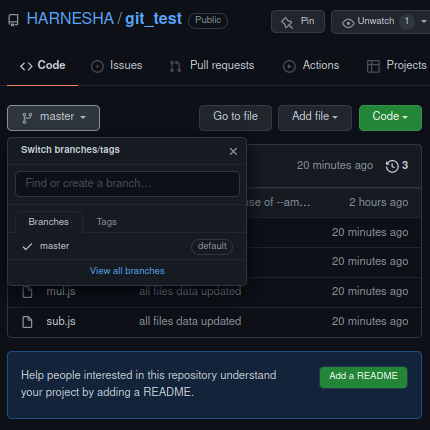
Let’s create a branch with the name dev. To create a branch, first, we need to make sure we are inside the correct branch with the begin with. Right now, we are inside the *master* branch and you can verify that by looking at your terminal or by checking how many branches are present in the repository. The one with an asterisk is the branch you are currently in.

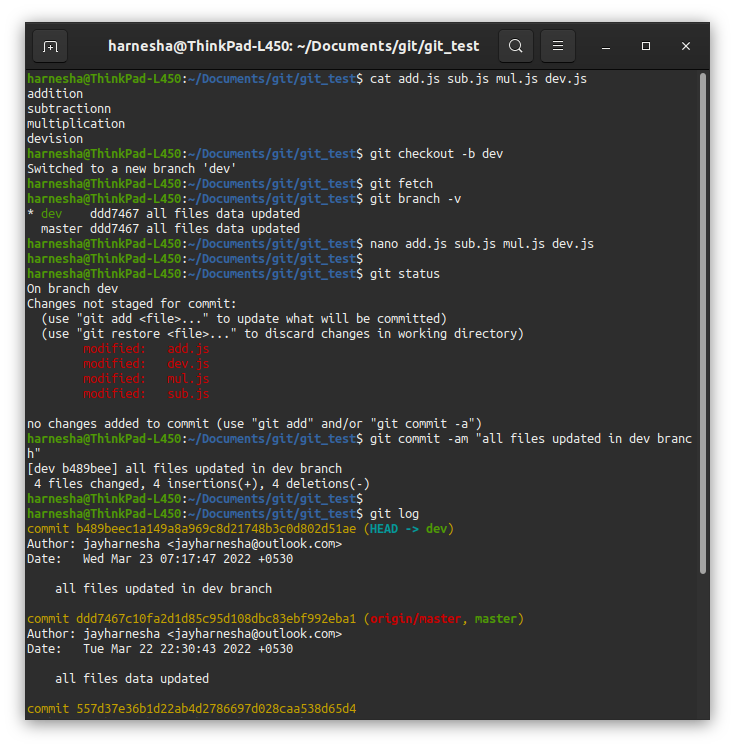


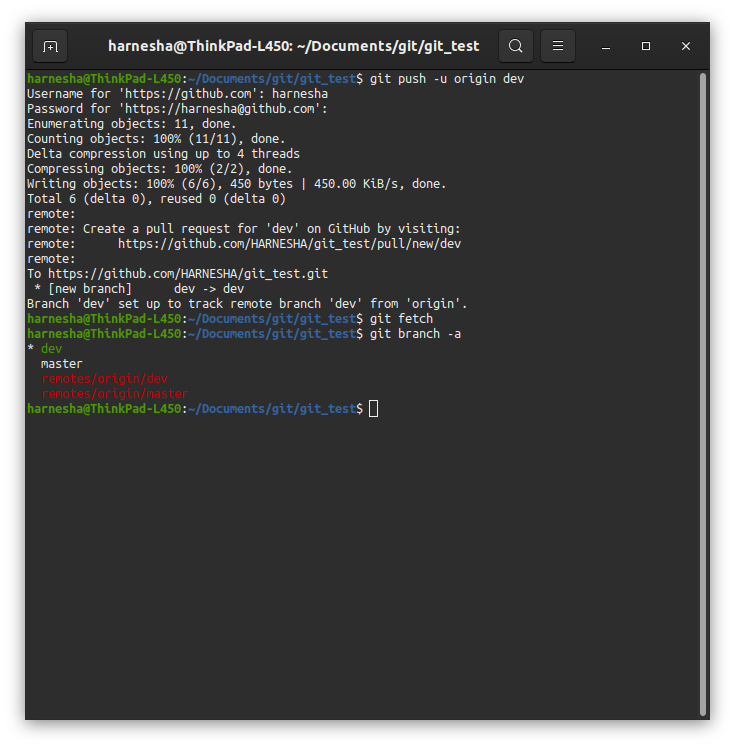
Create a commit in dev branch which will be associated with dev branch in local repo.

Create origin of dev branch in repo and push that repo to git hub.

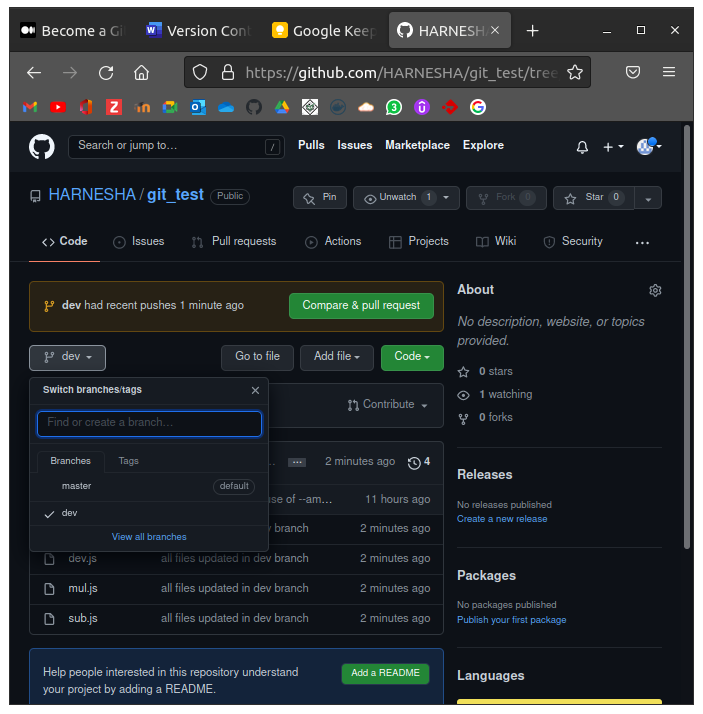






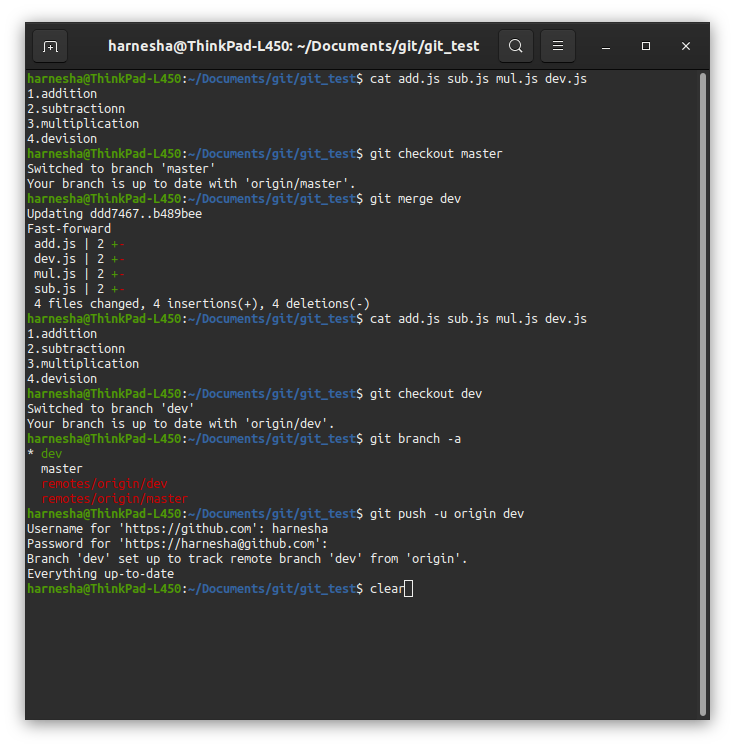


Verify uploaded code on github repo and also branch in which code is present.

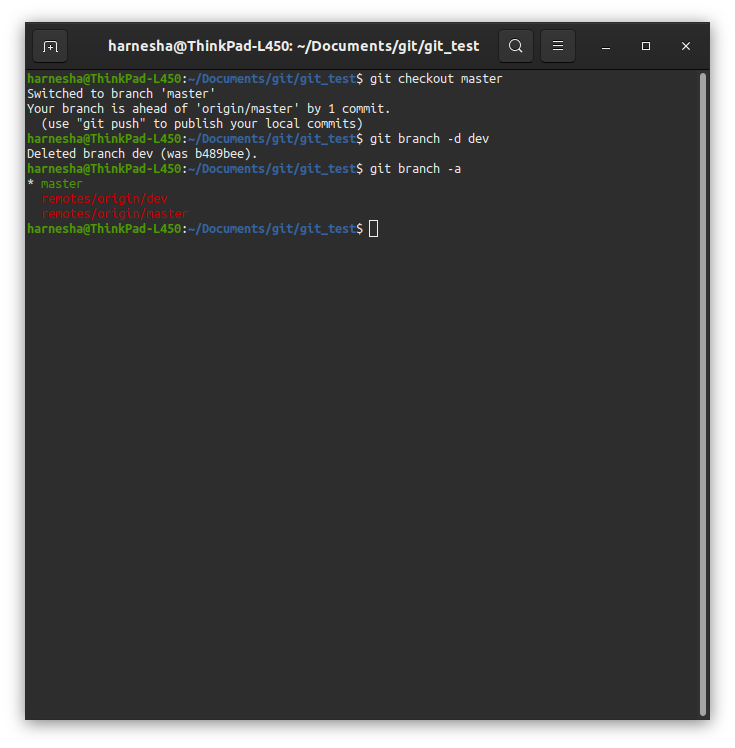


Update file data and checkout to master branch.

After that we can merge our dev branch updated files to out master branch.



After updating master branch we can remove branch if there is no user of further.



We have seen so far that if you are working with a team of people, then you should not touch the production branch which in our case is master. But what if you accidentally forgot to switch branch and made commits inside the master branch? You can’t just remove your commits using git reset and redo the work. That would be painful. In that case, we could use couple of techniques including **cherry-picking**.

